VIEDOC eLearning

Using JavaScript in Viedoc

1 Expressions

- An expression in Viedoc is a JavaScript function body written in ECMAScript 5.1 standards http://www.ecma-international.org/ecma-262/5.1
  http://www.w3schools.com/js/js_syntax.asp

  E.g.

```
return 2;
```

```
var a=2;
var b=3;
return a+b;
```

- Expressions can be written as **single statement**. Viedoc converts them into function body.

```
expression is converted to return (expression)
```

```
2 to return (2);
```

```
"hello"
```

```
2+2
```

```
2>3 to return (2>3);
```

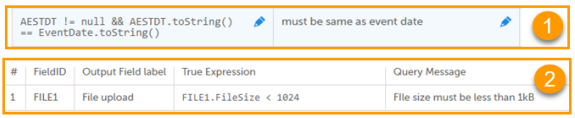
```
VSDT <= now()
```

  converts to

```
return ( VSDT <= now());
```

- **WARNING!** When using loops (`for, for/in, while, do/while`) consider the following:
  - Do not copy code (from internet) without reviewing it first and checking what it does.
  - Make sure that you don't have endless loops.
  - Consider browser compatibility when using javascript functions.
  - Avoid inline/recursive functions, as they could cause memory problems in the system, both on the client as well as on the server side.

2 Boolean expressions



- The boolean expressions are the expressions that return a boolean value, either `true` or `false`. They are used in the **Visibility** and **Validation**

  For JavaScript comparators and logical operators, you can refer to http://www.w3schools.com/js/js_comparisons.asp.

  - `true` - everything with a real value is `true`:

```
100
```

```
"Hello"
```

```
"false"

7 + 1 + 3.14

5 < 6
```

- `false` - everything without a real value is `false`:

```
0

Undefined

null

""
```

- **Date comparison**

  Dates in JavaScript are objects and the available comparators are > , >= ,
  < and <=.

  If you need to use == or !=, convert to string and compare, as in example (1) in the image, except for the case when checking for NULL. When checking for NULL, do not convert as this would lead to a change of the value during conversion from NULL to something different.

  Note! Remember to check for NULL before invoking any function on an object.

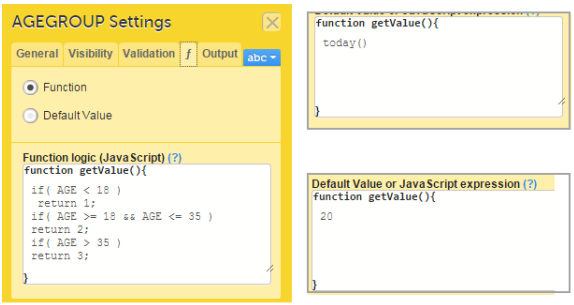
- **File properties**

  The metadata values of the file datatype can be accessed in expressions, as shown in the image at (2).

3 Value expressions

- The value expressions are those expressions that return a value.
  They are used in Viedoc for function and for the default value.

  Note! Expressions that returns a value to form item must match the data type as specified in data type table in the below section.



4 Data types

- The following table lists the Viedoc items together with their JavaScript types and default values.

Viedoc item	JavaScript type	Default value
Single line text	String	null
Number	Number	null
Date	Date	null
Time	Date	null
Paragraph text	String	null
Checkbox	Array of string/number*	[ ]
Radio button	String/Number*	null
Dropdown	String/Number*	null

File	Object with following members: ◦ FileName (string) - name of the uploaded file ◦ FileSize (number) - file size in bytes ◦ FileHash (string) - MD5 hash of the file content	null
Range	Object with following members: ◦ Lower (number) - the lower limit of the range ◦ LowerFormat (number) - the number of decimals used for the lower limit of the range ◦ Upper (number) - the upper limit of the range ◦ UpperFormat (number) - the number of decimals used for the upper limit of the range ◦ Comparator (string) - the comparator used to define the range. The available comparators are: • `InclusiveInBetween` - defines a range beween a lower and an upper defined limits. • `LessThan` • `LessThanOrEqualTo` • `GreaterThan` • `GreaterThanOrEqualTo` • `EqualTo`	null

**The item type for checkbox, radio button or dropdown is usually number, unless any of the items is not a number.*

5 Function

- Functions are evaluated and the resulting value is set to the item in the following scenarios:
  - Initialize form
  - During form edit, when any dependencies change.
  - When the form is upgraded while applying a revision.
  - When **Auto update** option is checked on a form and a cross-form dependant variable value is changed.

6 Default value

- The default value expressions are executed and the resulting value is set only when the form is initialized.

  Note! If there is visibility condition set on item with function or default value, whenever item becomes hidden, their value is reset to default value.

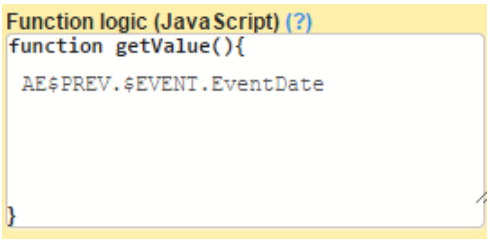
7 Context variables

- When an expression is evaluated in a form context, following variables are accessible.

Variable name	Data type	Default value
`{ItemDefId}`	As specified in data types table in section 4	As specified in data types table in section 4
`SubjectKey`	String	null only for add patient
`SiteSubjectSeqNo`	Number	Sequence number of the subject in the site (starts with 1)
`StudySubjectSeqNo`	Number	Sequence number of the subject in the study (starts with 1)

`SiteCode`	String	The site code as set in Viedoc Admin
`CountryCode`	String	Two letter ISO country code of the site
`StudyEventDefId`	String	The ID of the study event as specified in the study workflow in Viedoc Designer
`StudyEventType`	String	"Scheduled", "Unscheduled" or "Common"
`FormDefId`	String	The ID of the form as specified in Viedoc Designer > forms > settings.
`EventDate`	Date object	Current date for common events, all other current events date
`ActivityDefId`	String	The ID of the activity as specified in the study workflow in Viedoc Designer
`{ItemDefId}__format`	Number	Date types: 0. Date only 1. Date and time 2. Day not known 3. Month not known e.g. `ICDATE__format` Numeric with decimals `n:` precision/number of decimals

8 Cross form variables



- You can access any other form item in an expression.

  Syntax: `{EventDefId}.{FormDefId}.{ItemDefId}`

  E.g. A validation expression to check that the weight is > 65 for males and > 45 for females. Gender is collected in Patient Information and weight is collected in Demographics in each visit.

```
if (SCR.PI.GENDER == 'M')
return WEIGHT > 65;
else
return WEIGHT > 45;
```

  Notes!

  • An event is any event defined in workflow.
  • The JavaScript language definition allows space between keywords, but spaces inside cross form variable are not allowed, because, in order to provide values to execution context, Viedoc parses the whole expression for syntax as specified above.

- **Relative path**

  It is also possible to access cross form variable using relative path. Some special keywords are used in this case. Keywords that can be appended to `StudyEventDefId`. These keywords can also be used without `StudyEventDefId` to select any event.

`$FIRST`*n*	Select the first event, *n* is an optional index and it goes forward (meaning that on a timeline `$FIRST2` comes before `$FIRST3`)	`AE$FIRST.AEFORM.AEDATE`
		`$FIRST.DM.HEIGHT`
		`$FIRST2.DM.WEIGHT`

`$LAST`*n*	Select the last event, *n* is an optional index and it goes backwards (meaning that on a timeline `$LAST2` comes after `$LAST3`)	`AE$LAST.AEFORM.AEDATE`
`$PREV`*n*	Select the previous event, *n* is an optional index and it goes backwards (meaning that on a timeline `$PREV2` comes after `$PREV3`	`WEIGHT < $PREV.DM.WEIGHT + 10`

In addition

`$THIS`	Select the current context event

If a form appears in multiple activities in same event, `${ActivityDefId}` can be appended to select a specific form instance.

E.g. `WEIGHT <= $THIS.DM$MORNING.WEIGHT`

Here comes an example of using the optional indexer (*n* in the above table):
On the *Add patient* form, we have a text item that should have the latest non-blank value of another text item (called *NAME*) that is present on a form (called *PROFILE*) which is present on all scheduled (the first one called *START*) and unscheduled events. The function below can be used on the item of the *Add patient* form:

```
if($LAST.PROFILE.NAME != null) return $LAST.PROFILE.NAME;
if($LAST2.PROFILE.NAME != null) return $LAST2.PROFILE.NAME;
if($LAST3.PROFILE.NAME != null) return $LAST3.PROFILE.NAME;
if($LAST4.PROFILE.NAME != null) return $LAST4.PROFILE.NAME;
if($LAST5.PROFILE.NAME != null) return $LAST5.PROFILE.NAME;
if(START.PROFILE.NAME != null) return START.PROFILE.NAME;
return 'NOT SET';
```

This will allow for the item being saved blank for the last 4 events, or else fallback to the value of the item on the start event.

- **Cross Event Date**

  Accessing the date of another event uses same principle as any cross form variable, use a fixed form id `$EVENT` and item id `EventDate`.

`AESTDT >= BL.$EVENT.EventDate`	Start date must be after BL visit date

9 Math library

- The ECMAScript contains math objects that can be used for mathematical calculations.

10 Viedoc provided functions

-

Function	Description / Implementation
`date(dateString)`	Converts date string to JavaScript date object. The dateString must be in "yyyy-mm-dd" format
`today()`	Returns current date in Viedoc Clinic
`now()`	Returns current date and time in Viedoc Clinic
`addDays (date, days)`	Add days to the date object

`age (fromDate, toDate)`	`function age(DMDOB, DMIC) {`
	`  if (!DMIC || !DMDOB)`
	`  return null;`
	
	`  var ageMilli = DMIC - DMDOB;`
	
	`  var ag = ageMilli / 1000 / 3600 / 24 / 365.25;`

	return ag; }

bmi (weightInKg, heightInCM)	function bmi(weight, height) {
	if (weight <= 0 || height <= 0)
	return null;
	var finalBmi = weight / (height / 100 * height / 100);
	
	return finalBmi;
	}

days (startDate, endDate)	function days(startDate, endDate) {
	if (!startDate || !endDate)
	return null;
	
	var oneDay = 24 * 60 * 60 * 1000; // hours*minutes*seconds*milliseconds
	
	startDate = date(startDate);
	endDate = date(endDate);
	
	var diffDays = Math.round((startDate.getTime() - endDate.getTime()) / (oneDay));
	
	return diffDays;
	}

hours(startDateTime, endDateTime)	function hours(startDateTime, endDateTime) {
	if (!startDateTime || !endDateTime);
	return null;
	
	var oneHour = 60 * 60 * 1000; // minutes*seconds*milliseconds
	
	var diffHours = Math.round((startDateTime.getTime() - endDateTime.getTime()) / (oneHour));
	
	return diffHours;
	}

minutes(startDateTime, endDateTime)	function minutes(startDateTime, endDateTime) {
	if (!startDateTime || !endDateTime);
	return null;
	
	var oneMinute = 60 * 1000; // seconds*milliseconds
	
	var diffMinutes = Math.round ((startDateTime.getTime() - endDateTime.getTime()) / (oneMinute));
	
	return diffMinutes;
	}

- ***Array.contains* function**

  `[].contains( x )` is a special function used to check if an item is present in array.

  Examples:

  - With checkboxes:

    `INCL.contains(1) || EXCL.contains(2)`

  - Skip validation in all 3 activities:

    ```
    var skipActivities = ['V1A1', 'V2A1', 'V3A1'];
    ```

```
if ( skipActivities.contains( ActivityDefId ) )
   return true;;
...
```

11 Debugging your expression

- To debug you can use one of the following in your expression:
  - `debugger;` statement.
  - `console.log('something');`

  The above statements will have effect only when opening the browser's developer tools while entering data to the respective form in Viedoc Clinic.

  For more information about debugging please refer to http://www.w3schools.com/js/js_debugging.asp.

  Notes!

  • When debugging, you cannot use single line statements.
  • You cannot debug the visit/activity visibility expression as they are run on server.

12 Validation

- During **Save changes** and **VALIDATE** operations in Viedoc Designer, the expressions are validated using a compiler, which would find most of the errors. However, since JavaScript is a dynamic language, not everything can be validated. For example, `AGE.foo ()` will not throw an error, because `AGE` is a variable in a form and the compiler does not know its type.

  Note! The designer must test the expression in all the possible paths using either preview or Viedoc Clinic.